64-041 Übung Rechnerstrukturen

Aufgabenblatt 5 Ausgabe: 16.11., Abgabe: 23.11. 24:00

Gruppe	
Name(n)	Matrikelnummer(n)

Aufgabe 5.1 (Punkte 8+2+5)

UTF-8: Die ISO-8859-1 Codierung benutzt 8 Bit für jedes enthaltene Zeichen. Die direkte Codierung der basic-multilingual Plane von Unicode (Java Datentyp `char`) verwendet pro Zeichen 16 Bit, während die UTF-8 Codierung Vielfache von 8 Bit benutzt.

(a) Wir betrachten einen deutschsprachigen Text mit insgesamt 750 000 Zeichen. Wir nehmen die folgenden Wahrscheinlichkeiten für die Umlaute an, andere Sonderzeichen kommen nicht vor:

Ä/ä	Ö/ö	Ü/ü	ß
0,56%	0,287%	0,616%	0,307%

Wie viele Bytes belegt dieser Text bei Codierung nach ISO-8859-1, in direkter Unicode Darstellung und in UTF-8?

(b) Wir betrachten einen chinesischen Text mit insgesamt 750 000 Schriftzeichen. Im aktuellen Unicode-Standard sind für die CJK-Symbole (chinesisch, japanisch, koreanisch) die Bereiche von `U+3400` bis `U+4DB5` und `U+4E00` bis `U+9FD5` reserviert.

Wie viele Symbole sind das?

(c) Wie viele Bytes belegt der chinesische Text bei direkter Unicode Darstellung und bei Codierung als UTF-8?

Aufgabe 5.2 (Punkte 5+5+5+5)

Shift-Operationen statt Multiplikation: Ersetzen Sie die folgenden Berechnungen *möglichst effizient* durch eine Folge von Operationen: <<, +, -. Nehmen Sie für die Variablen x und y den Datentyp `int` (32-bit Zweierkomplementzahl) an.

(a) $y = 6 \times x$

(b) $y = 30 \times x$

(c) $y = -56 \times x$

(d) $y = 62 \times (x + 4)$

Aufgabe 5.3 (Punkte 5+5+7+7+11)

Logische- und Shift-Operationen: Realisieren Sie, die folgenden Funktionen als *straightline*-Code in Java, das heißt ohne Schleifen, If-Else Abfragen oder den ternären Operator `.. ? .. : ...` Außerdem dürfen nur einige der logischen und arithmetischen Operatoren benutzt werden:

```
! ~ & ^ | + << >> >>>
```

Alle Eingabeparameter und Rückgabewerte sind jeweils (32-bit) Integerwerte.

(a) `bitNor(x,y)` Diese Funktion soll das bitweise NOR liefern: $\overline{x_i \vee y_i}$. Als Operatoren dürfen nur `&` und `~` (AND, Negation) benutzt werden.

(b) `bitXnor(x,y)` Diese Funktion soll die XNOR-Verknüpfung (Äquivalenz) realisieren: $x_i \equiv y_i$. Als Operatoren dürfen nur `|` und `~` (OR, Negation) benutzt werden.

(c) `getByte(x,n)` Diese Funktion soll das, durch `n` angegebene Byte ($0 \leq \mathtt{n} \leq 3$) aus dem Wert `x` extrahieren.

(d) `rotateRight(x,n)` Die Funktion soll den in Java nicht vorhandenen Rotate-Right Operator für `x` nachbilden. Für das zweite Argument `n` gilt: $0 \leq \mathtt{n} \leq 31$.

(e) `abs(x)` Der Absolutwert (Betrag) von `x`. Welchen Wert liefert ihre Funktion für den Eingabewert -2^{31}? Beschreiben Sie, wie Ihre Lösung funktioniert.

Aufgabe 5.4 (Punkte 10)

Base-64 Codierung: Wie in der Vorlesung skizziert, werden bei der Base-64 Codierung jeweils drei 8-bit Eingangswerte durch vier 6-bit Ausgangswerte ersetzt, die dann zur Datenübertragung als (7-bit) ASCII-Zeichen codiert werden.

Beschreiben Sie durch logische- und Schiebe-Operationen, wie bei der Decodierung aus den vier Eingabezeichen `a1...a4` (hier schon als Integer Zahlen), die drei 8-bit Ausgangswerte `b1...b3` berechnet werden. Vervollständigen Sie dazu die Ausdrücke `b...` im nachfolgenden Java-Code.

```
int a1, a2, a3, a4;          // vier Zeichen, Wertebereich je 0..63

int b1 = ?
int b2 = ?
int b3 = ?

...
```

Aufgabe 5.5 (Punkte 10)

Codierung: Die 26 Großbuchstaben des Alphabets sollen in einem zyklisch-einschrittigen Binärcode „durchgezählt“ werden. Entwickeln Sie so einen Code mit dem, in der Vorlesung vorgestellten, rekursiven Verfahren.

Aufgabe 5.6 (Punkte 10)

Codierung: Erläutern Sie, warum es keinen zyklisch-einschrittigen (Binär-) Code mit ungerader Zahl von Codewörtern geben kann.